1. **An invoice contains a collection of purchased items. Should that collection be implemented as a list or set? Explain your answer.**

In this situation, I would suggest using a list rather than a set. This is because lists are collections

that maintains the order of its elements. More importantly, lists will accept duplicate entries. On

the other hand, sets are unordered collections of unique elements, meaning that they do not accept duplicate entries. In this scenario, a list would work better because the order in which the items were purchased might matter on the invoice. Additionally, if the buyer purchases more than one of an item, the invoice would need to be able to accept duplicate items to increase the quantity. Because lists can accept duplicates and sets can’t a list is the better choice here.

1. **Consider a program that manages an appointment calendar. Should it place the appointments into a list, stack, queue, or priority queue? Explain your answer.**

For an appointment calendar, I would suggest using a priority queue. This is because, generally, appointments will occur in a first-in first-out manner. This is because people who show up for appointments earlier or make appointments earlier often are seen for their appointments before others. However, if an appointment is especially urgent, it can be given priority in the queue. Because priority queues have the first-in first-out order and can prioritize certain elements, they would be the best choice for an appointment calendar.

1. **One way of implementing a calendar is as a map from date objects to event objects. However, that only works if there is a single event for a given date. How can you use another collection type to allow for multiple events on a given date?**

You can allow for multiple events on a given date by using lists within the map. More specifically, you can use an ArrayList of events to store all events for a certain date. Then, you can create the map using dates and the ArrayList of events rather than just date objects and event objects.

1. **Explain what the following code prints. Draw a picture of the linked list after each step.  
    LinkedList<String> staff = new LinkedList<>();  
   staff.addFirst("Harry");  
   staff.addFirst("Diana");  
   staff.addFirst("Tom");  
   System.out.println(staff.removeFirst());  
   System.out.println(staff.removeFirst());  
   System.out.println(staff.removeFirst());**

This code will print the following:

Tom

Diana

Harry

Below is a diagram of the LinkedList after each step:

1. staff = []
2. staff = [Harry]
3. staff = [Diana, Harry]
4. staff = [Tom, Diana, Harry]
5. staff = [Diana, Harry]
6. staff = [Harry]
7. staff = []
8. **Explain what the following code prints. Draw a picture of the linked list after each step.  
    LinkedList<String> staff = new LinkedList<>();  
   staff.addFirst("Harry");  
   staff.addLast("Diana");  
   staff.addFirst("Tom");  
   System.out.println(staff.removeLast());  
   System.out.println(staff.removeFirst());  
   System.out.println(staff.removeLast());**

The code will print the following:

Diana

Tom

Harry

Below is a diagram of the LinkedList after each step:

1. staff = []
2. staff = [Harry]
3. staff = [Harry, Diana]
4. staff = [Tom, Harry, Diana]
5. staff = [Tom, Harry]
6. staff = [Harry]
7. staff = []
8. **You are given a linked list of strings. How do you remove all elements with length less than or equal to three?**

You would first create a new iterator object for the linked list. Then, you would use a while loop and the hasNext method for iterators to go through each element of the linked list. In the while loop, you could use an if statement and the iterator’s next method to see if the next element of the linked list has a length of 3 or less. If it does, then use the iterator’s remove method to remove the element. This loop will go through all elements of the linked list, check their length, and remove them if they are not at least 4 characters long. Here’s the while loop that could be used:

while(iter.hasNext()){

if(iter.next().length() <= 3){

iter.remove();

}

}

1. **Suppose you need to organize a collection of telephone numbers for a company division. There are currently about 6,000 employees, and you know that the phone switch can handle at most 10,000 phone numbers. You expect several hundred lookups against the collection every day. Would you use an array list or a linked list to store the information?  Why (not)?**

For this situation, you would want to use an ArrayList to store the information. This is because any ArrayList element can be accessed using the ArrayList index, so the lookup process for hundreds of numbers would much quicker. However, for a LinkedList, you must use an iterator to move through each entry. This process would greatly slow down the lookup process. Finally, retrieving data from an ArrayList is also much easier than retrieving from a LinkedList, so I would use an ArrayList in this situation.

1. **Suppose the strings "A" . . . "Z" are pushed onto a stack. Then they are popped off the stack and pushed onto a second stack. Finally, they are all popped off the second stack and printed. In which order are the strings printed?**

The strings will be printed in forwards order. In other words, they will be printed from A to Z. This is because when they are first popped off the stack, they will be popped from Z to A. So, when they are popped from the second stack, they will be popped and printed from A to Z.

1. **How can you compute the union and intersection of two sets, using some of the methods that the java.util.Set interface provides, but without using an iterator?**

In order to compute the union of two given sets, you can use the addAll method provided by the set interface. This method adds any elements of a given set (that are not already present) to the set the method is being called for (ex: set1.addAll(set2) gives union of set1 and set2 by adding any elements of set2 that aren’t already present to set1). Additionally, you can use the retainAll method provided by the set interface to compute the intersection of two sets. This method only retains the elements of the set which the method is called for if they are also in the given set (ex: set1.retainAll(set2) creates intersection of set1 and set2 by removing any elements from set1 that are not in set2).

1. **Can a map have two keys with the same value? Two values with the same key?**

Yes, you can have a map that has two separate keys with the same value. For example, if I make a Map for relationships where the keys and values are both strings, I could assign the key “Mom” the value of “Julie” and I could also assign the key “Friend” the value of “Julie”. However, you cannot have two values with the same key. This is because if you try to use the put method to add another value for the same key, it will simply modify the existing value for that key. For example, if I then tried to assign the key “Mom” the value of “Anne”, then “Mom” would simply change from “Julie” to “Anne”. Additionally, if you tried to assign one key multiple values in the same line, you would get an error.